

Final Report: Developing Semantically-aware and Web-enabled KNSG Application Framework

October 30, 2011

PI: Danny Powell
danny@ncsa.illinois.edu

Co-PI: Jong S. Lee
jonglee@ncsa.illinois.edu

National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
1205 W. Clark St., Urbana, IL 61801, USA

Final Report: Developing Semantically-aware and Web-enabled KNSG Application Framework

Table of Contents

1	Introduction.....	1
2	Technology, Development Tools and Resources.....	2
2.1	HPC Resources.....	2
2.2	Apache Maven.....	2
2.3	Google Web Toolkit.....	2
2.4	Java SE (EE) 1.6.....	3
2.5	RESTful Web Services.....	3
2.6	Apache Lucene.....	3
2.7	Semantic Content Management Middleware	3
2.8	Semantic Workflow.....	4
2.9	Publishing Active Workflow (PAW)	4
2.10	Gondola: HPC Job Submission and Monitoring	5
3	Architecture.....	5
3.1	System Architecture	6
3.2	Security Architecture.....	7
4	System Settings and Installations.....	9
4.1	Semantic Content Repository (Tupelo Repository)	9
4.2	Semantic Content Repository Server	10
4.3	Cyberintegrator Engine Server.....	11
4.4	Gondola Server.....	12
4.5	eAIRS Web Application.....	13
4.5.1	Configuring Tomcat.....	14
4.5.2	Deploying WAR	14
4.5.3	Configuring eAIRS	14
5	User Guide	15
5.1	Gondola	15
5.2	How to Create and Export a New Workflow	17
5.3	eAIRS Web Application.....	24

6	Future work.....	27
7	Statement of Expenses	28
	Appendix A: XSD Schema definition for Gondola	29
	Appendix B: An example Context.xml.....	33

Figures

Figure 1. Cyberintegrator Editor	4
Figure 2. Published Workflow	5
Figure 3. Enhanced KNSG Application Framework Architecture	6
Figure 4. Implemented System Architecture	7
Figure 5. Security System Architecture	8
Figure 6. System Settings	9
Figure 7. New Workflow Dialog	17
Figure 8. Tools View	17
Figure 9. Tools Execution View for eAIRS Parameter Tool	18
Figure 10. Workflow Graph with eAIRS Parameter Tool	18
Figure 11. Tool Execution View for Trebuchet Tool	19
Figure 12. Workflow Graph with eAIRS Parameter Tool and Trebuchet	19
Figure 13. Tool Execution View for eAIRS CFD Tool	21
Figure 14. Complete Workflow Graph	21
Figure 15. Workflow Information	22
Figure 16. Export Workflow Dialog	22
Figure 17. Specify Name and Location of Exported Workflow	23
Figure 18. Import Dialog	23
Figure 19. Workflow to Import into Repository	24
Figure 20. Login Page	24
Figure 21. Create a Simulation Page	25
Figure 22. Launch Simulation Page	25
Figure 23. Mesh Generator (Viewer)	26
Figure 24. Monitoring and Browsing Page	26
Figure 25. Search Page	27

Tables

Table 1. Gondola REST API	15
---------------------------------	----

1 Introduction

National Center for Supercomputing Applications (NCSA) and Korea Institute of Science and Technology Information (KISTI) have developed a prototype non-domain-specific platform called KNSG (KISTI-NCSA Science Gateway) Application Framework (KNSG-AF) for building domain-specific HPC applications. The Application Framework provides a core set of reusable components for building new applications with Eclipse RCP¹.

As a continuing work, this project titled, “Developing Semantically-aware and Web-enabled KNSG Application Framework”, achieved the following goals in order to enhance the KNSG Application Framework:

1. Enable KNSG Application Framework to be semantically-aware
2. Enable KNSG Application Framework to develop web applications
3. Enable Middleware Services to support multi-users and heavy concurrent connections

As a result of the project, NCSA and KISTI have enhanced KISTI-NCSA Science Gateway (KNSG) Application Framework by adding semantic content management and extending the framework to support web application development. Note that Dr. Dukyun Nam’s visit to NCSA was a tremendous help and enabled an efficient collaboration among members of this project.

The enhancement was facilitated through the inclusion of underlying technologies such as PAW (Publishing Active Workflow), Tupelo (Semantic Content Management Middleware), Cyberintegrator (Semantic Workflow Engine and Management) and Gondola (lightweight version of PTPFlow). The design of Gondola, which is a by-product of this project, allows it to be used independently of the framework and provide simple and lightweight middleware for HPC job submission and monitoring via RESTful service.

The use case chosen to demonstrate the capabilities of the enhanced KNSG Application Framework is the eAIRS web application. This version of eAIRS and KNSG Application Framework will focus on providing an easy-to-use web user interface (UI) for generating Mesh files, setting up the eAIRS CFD Workflow, launching it to HPC via Gondola, monitoring the jobs and visualizing the results.

In this report, Section 2 describes the development tools, technology and resources needed for the project. Section 3 explains the overall architecture and implemented system architecture. Section 4 describes how to set up the implemented system. Section 5 contains user guides and tutorials. Finally, Section 6 discusses potential future work to extend and enhance the framework.

Note that all the documents in this report are available at the project website,
<https://wiki.ncsa.illinois.edu/display/KNSG/Home>.

¹ <http://www.eclipse.org/home/categories/rcp.php>

2 Technology, Development Tools and Resources

This section describes the technology, development tools and resources for the development of this project.

2.1 HPC Resources

This project uses the HPC resources available through Xsede². Specifically, Ranger and Blacklight were used for the development.

- TRACC Ranger³
 - Model: AMD Opteron
 - OS: Linux (CentOS)
 - Processors: 62976
 - Nodes: 3936
 - Memory: 123 TBytes
 - Peak Performance: 579.40 TFlops
 - Disk: 173 TBytes
 - SGE
- PSC Blacklight⁴
 - Model: SGI UV 1000 cc-NUMA
 - OS: SuSE Linux
 - Processors: 4096
 - Nodes: 512
 - Memory: 32 TBytes
 - Peak Performance: 37.20 TFlops
 - Disk: 150 TBytes
 - PBS

2.2 Apache Maven

Since the development requires components that are developed independently by multiple users, Apache Maven⁵ is utilized to manage the dependency among components and builds. Currently, CET group, NCSA, is maintaining a Nexus⁶ server Maven repository that manages all components developed at NCSA.

2.3 Google Web Toolkit

Google Web Toolkit⁷ (GWT) 2.4 is used for developing widgets and web applications. In addition, GWT plugin for Eclipse, GWT designer, and gwt-maven are used in the GWT

² <http://www.xsede.org/>

³ <http://www.tacc.utexas.edu/resources/hpc>

⁴ <http://www.psc.edu/machines/sgi/uv/blacklight.php>

⁵ <http://maven.apache.org/>

⁶ <http://nexus.sonatype.org/>

⁷ <http://code.google.com/webtoolkit/>

development environment. In the development of the web application, GWT widgets follow the MVP (Model-View-Presenter)⁸ architecture so that the widgets can be reused easily.

2.4 Java SE (EE) 1.6

The KNSG application framework is built against Java SE or Java EE 6.

2.5 RESTful Web Services

The project developed multiple RESTful Web Services⁹ (e.g. Gondola) which are web services implemented using HTTP and REST (REpresentational State Transfer)¹⁰ architecture. In order to build the RESTful web services, Apache Wink¹¹ is used on both server side and client side. The Apache Wink Server module is an implementation of JAX-RS v1.1 specification (JSR 311)¹² which is Java API for RESTful Web Services. The JAX-RS v1.1 is an official part of Java Enterprise Edition (EE) 6.

Note that Jackson Java JSON-processor¹³ is utilized for JSON along with Apache Wink.

2.6 Apache Lucene

In order to support the search capability, Apache Lucene¹⁴ is used. It is a full-text search engine library. It's widely used a full-text indexing engine. Currently, eAirs indexes the title and description of simulation when a user submitted a job by Apache Lucene.

2.7 Semantic Content Management Middleware

In order to manage the semantic content such as workflows and track provenance, semantic content management middleware is needed to manage, store and query RDF triples. Tupelo¹⁵, developed by NCSA, is utilized for this project.

Tupelo is a data and metadata management system based on semantic web technologies. Tupelo provides a variety of generic utilities for managing data and metadata using both best-of-breed semantic database implementations such as Jena and Sesame, as well as simple storage technologies such as flat files. Note that this project stores RDF triples in a MySQL database. Tupelo makes data and metadata portable across a variety of Contexts and deployment scenarios, including desktop applications, web-based applications, and more complex distributed architectures. Its use of global identification and explicit semantics means that metadata created and managed with Tupelo can be easily exported and used by a wide variety of RDF-aware tools and technologies.

⁸ <http://code.google.com/webtoolkit/doc/latest/DevGuideMvpActivitiesAndPlaces.html>

⁹ http://en.wikipedia.org/wiki/Representational_state_transfer#RESTful_web_services

¹⁰ http://en.wikipedia.org/wiki/Representational_state_transfer

¹¹ <http://incubator.apache.org/wink/>

¹² <http://jcp.org/en/jsr/detail?id=311>

¹³ <http://jackson.codehaus.org/>

¹⁴ <http://lucene.apache.org/>

¹⁵ <https://opensource.ncsa.illinois.edu/confluence/display/TUP/Tupelo>

Tupelo is designed for managing large-scale, complex scientific data and metadata collections. It is also suitable for more conventional digital libraries containing Dublin Core and other standard digital library metadata schemas. Its RDF-based metadata framework can support a plethora of schemas, from simple, flat-namespace schemas such as Dublin Core, to hierarchical models derived from XML Schema, to more web-like models derived from RDF variants such as RSS.

2.8 Semantic Workflow

Analyses in KNSG-AF are expressed as workflows. The provenance tracking is done via semantic workflow management. Cyberintegrator (CI) and Cyberintegrator Engine¹⁶ are built on Tupelo and manage, store, query and execute the semantic workflow in KNSG-AF.

Cyberintegrator provides is a user friendly user interface to several middleware software components that (1) enable users to easily include tools and data sets into a software/data unifying environment, (2) annotate data, tools and workflows with metadata, (3) visualize data and metadata, (4) share data and tools using local and remote context repository, (5) execute step-by-step workflows during scientific explorations, and (6) gather provenance information about tool executions and data creations.

The middleware software components that make up Cyberintegrator Engine: (1) provide multiple plug-in modules called executors for running external tools such as Matlab scripts and Java codes and an extension point to add new executors, and (2) Tupelo data and metadata archiving system based on the Resource Description Framework (RDF) metadata model.

The Cyberintegrator editor is built on top of the Eclipse Rich Client Platform (RCP) to achieve its full functionality of re-configurable user interface and plug-and-play architecture critical for easy deployment of Cyberintegrator application and adding enhancements to the system.

2.9 Publishing Active Workflow (PAW)

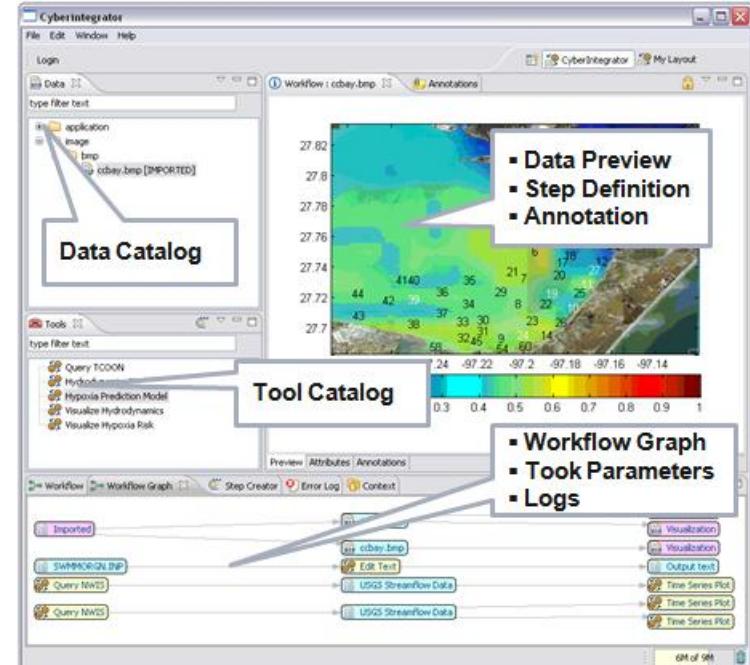


Figure 1. Cyberintegrator Editor

¹⁶ <https://opensource.ncsa.illinois.edu/confluence/display/CBI/Cyberintegrator+Home>

PAW (Publishing Active Workflow; previously known as Digital Synthesis Environment¹⁷) allows the user to group workflows into a scenario, expose certain parameters of the workflow, and enables the re-execution of workflows on a per user basis that create custom runs of the workflow for each user. Discussion detailed description of PAW can be found in Section 3.

The published workflow has the mapping of the UI widget to input/output parameters along with visibility of the parameters to users as shown Figure 2.

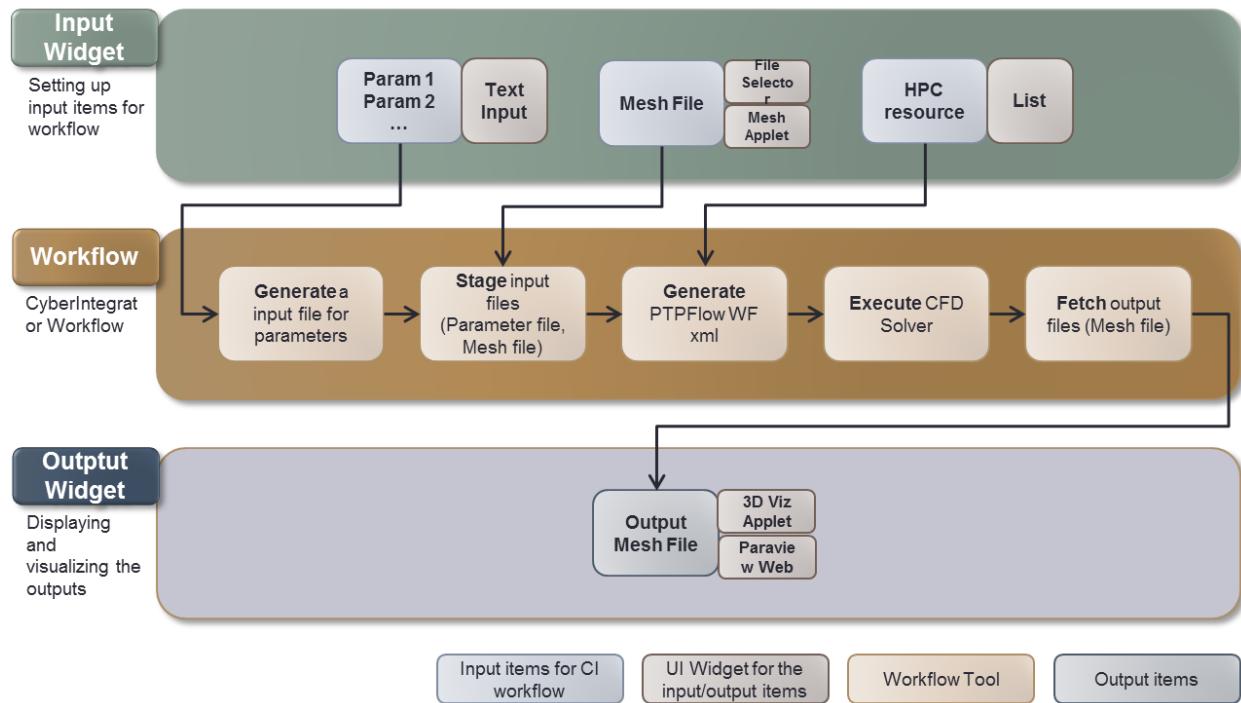


Figure 2. Published Workflow

2.10 Gondola: HPC Job Submission and Monitoring

The previous project utilized PTPFlow as the middleware and service for HPC job submission and monitoring. Although PTPFlow supports its own workflow mechanism, it was heavier weight than the project required. Therefore, Gondola, a lightweight and flexible service for HPC job submission and monitoring based on PTPFlow, was developed. It provides RESTful endpoints to submit and monitor jobs and can be utilized in other projects requiring job submission and monitoring facilities by using the API described in Section 5. The Gondola utilized the source code of PTPFlow mostly in terms of handling X509 certificate, GridSSH, etc. The detail user guide is described in Section 5.

3 Architecture

¹⁷ <http://isda.ncsa.uiuc.edu/DSE2/>

With the technology, development tools and resources described in Section 2, the architecture of enhanced KNSG Application Framework has been developed as shown in Figure 3. The diagram expresses the architecture in terms of managing, publishing, executing, and searching semantic content and shows how domain-specific scientific applications can be built on these middleware stacks and external services and resources. eAirs represents how applications can use the enhanced KNSG-AF and is built on this architecture.

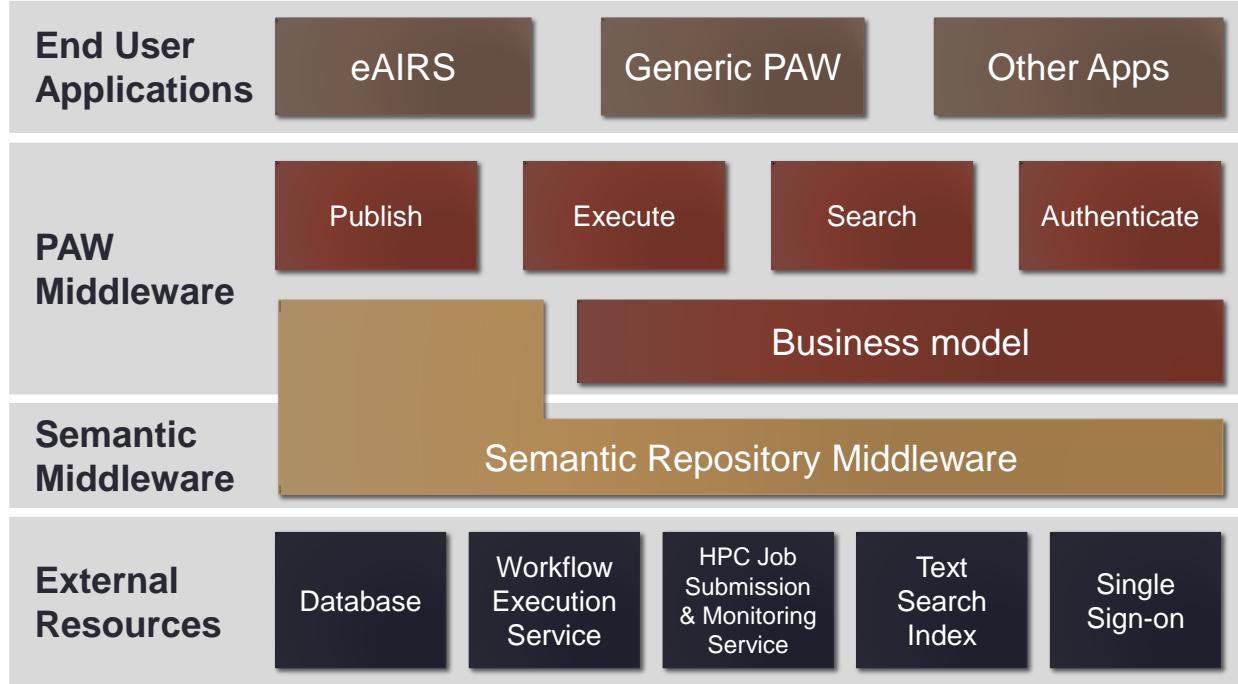


Figure 3. Enhanced KNSG Application Framework Architecture

PAW middleware provides the capabilities to publish, search, execute and monitor workflows. Moreover, it provides the mapping between input/output parameters and UI widgets so that users can use the published workflow easily and authors can expose only the parameters they want users to manipulate. The PAW middleware is built on Semantic Content Management middleware (Tupelo) and consumes external resources such as data services (MySQL), workflow execution service (CI engine), HPC job submission and monitoring service (Gondola), text search index service (Lucene), and single sign-on service (MyProxy).

3.1 System Architecture

The architecture has been implemented as the KNSG-AF system shown in Figure 4. In a typical use case, the following actions are occurring in this system:

1. The webapp built on PAW requests/receives list of workflows (solvers) to/from semantic content repository
2. The user selects the workflow (solver) and sets up all parameters
3. The user submits (or execute) the computation (or simulation)
4. The webapp submits the URI of workflow and all parameters
5. CI service gets the workflow by URI from the repository

6. CI service creates a copy of workflow and populates the parameters to the workflow
7. CI service executes the workflow
 - a. Stages the input files by GridFTP tools
 - b. Submit the workflow xml to Gondola service
8. CI service monitors the status of job and returns the status to the webapp.

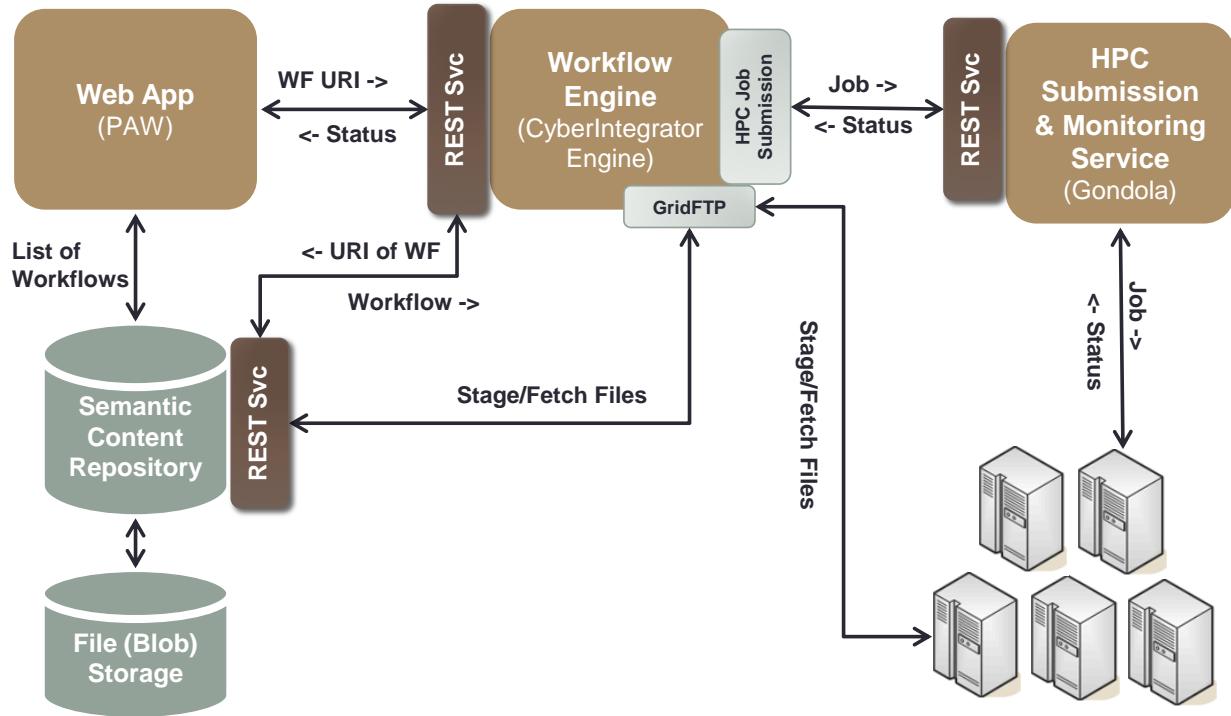


Figure 4. Implemented System Architecture

The web application is developed using PAW widgets. The widgets are automatically generated and populated in the Web UI based on the published workflow and the parts of the workflow the author exposes to the user. The workflow execution engine (CI Engine) has two tools that enable the system to submit jobs to HPC: 1) Gondola tool and 2) GridFTP tool. The Gondola tool is the REST client that uses Gondola RESTful service to submit the job and monitor the status of it. The GridFTP tool is used to stage in/out the files to/from HPC. All those activities are recorded in the Semantic Content Repository. In addition, the repository stores the files (or Blobs) to separate file storage server. This project uses mounted file system as file storage.

3.2 Security Architecture

The system has the security architecture as shown in Figure 5. The flow of logic in this system in terms of security is as follows:

1. The user submits username/password to the web server
2. The web server obtains Proxy Certificate from MyProxy (via MyProxy API) and stores it in the semantic content repository. The authentication for MyProxy can be set up with LDAP service as shown in the diagram.

3. CI workflow (or tools) obtains Proxy Certificate in order to submit job to HPC and stage files from/to HPC via GridFTP.

Note that all communications within green-dotted region will be trusted. This region needs to be implemented more securely if those services in the region need to be served in different domains.

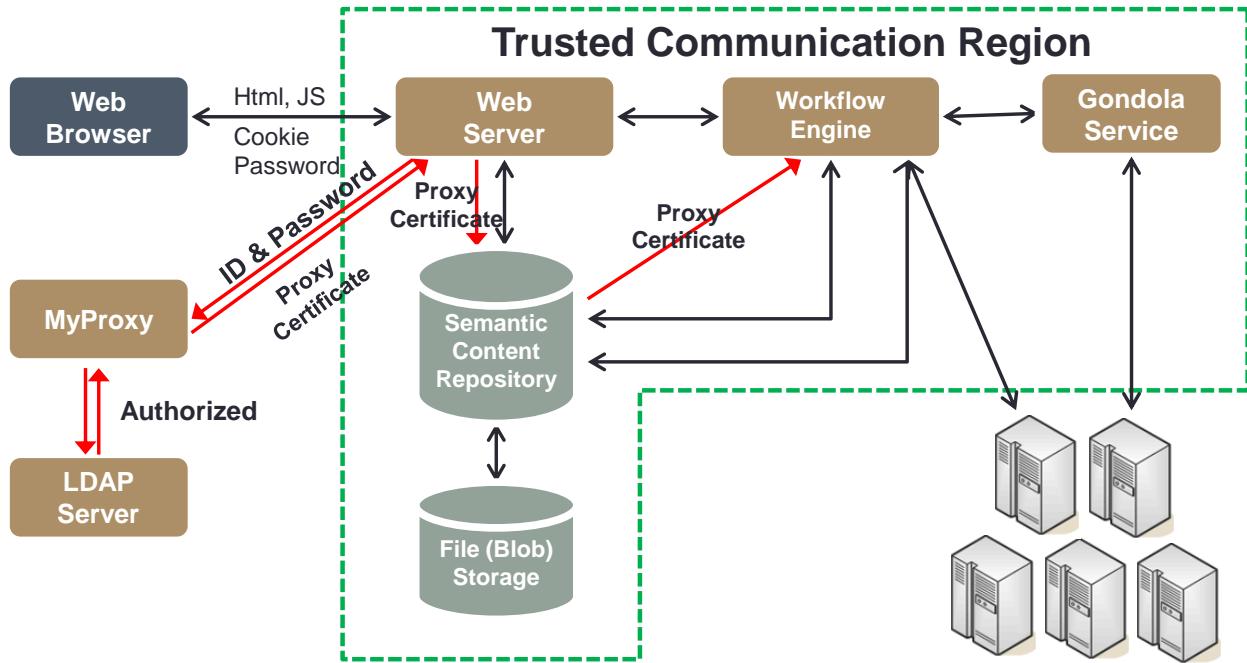


Figure 5. Security System Architecture

The Proxy X509 certificates have a limited lifetime. Currently, the initial lifetime is 1 hour. However, the certificate needs to be renewed when the job is running longer than 1 hour. In order to renew the certificate, there is a daemon service that checks the repository and renews the certificate with the following criteria:

- 1) If the user is logged-in, renew the certificate
- 2) If there are workflows (or jobs) running, renew the certificate

Otherwise, the daemon destroys the certificate.

4 System Settings and Installations

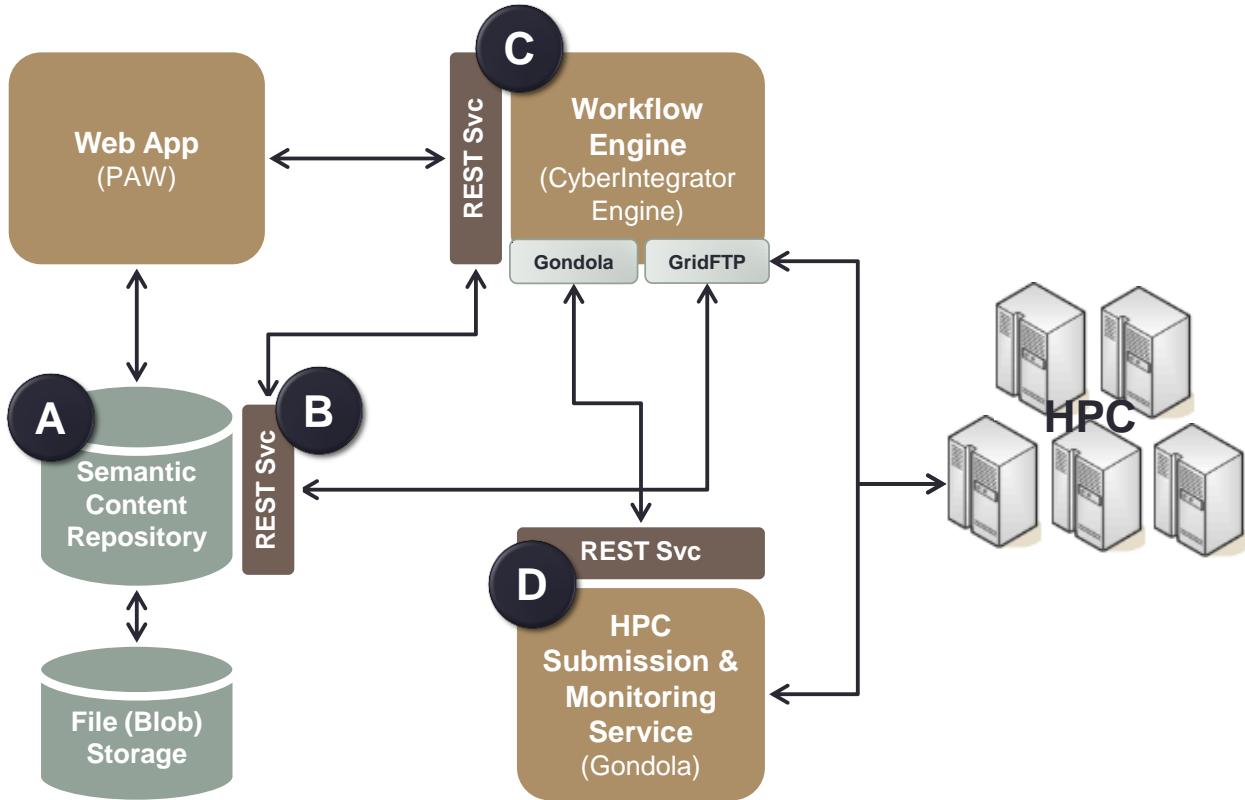


Figure 6. System Settings

The enhanced KNSG-AF requires users to setup the following services: (Figure 6 shows which service plays a role in the system architecture)

- Semantic Content Repository (Tupelo)
- Semantic Content Repository Context (Tupelo Context) Server (RESTful service)
- Cyberintegrator Engine Server (RESTful service)
- Gondola (RESTful service)

4.1 Semantic Content Repository (Tupelo Repository)

Tupelo supports various relational databases, such as MySQL, PostgreSQL, etc., as a triple storage. This project uses MySQL as a semantic content repository. The following steps create a database with Tupelo repository schema: (Note that the commands below are tested on the Linux platform)

- Acquire the SQL file for database schema setting:

```
wget --no-check-certificate -q -O mysql.sql \
https://opensource.ncsa.illinois.edu/confluence/download/attachments/589837/mysql.sql
```

2. Replace database name (db_name), user name (db_user) and user password (db_pass) and create a new SQL file (kisti.sql)

```
sed -e "s#@DB_SCHEMA@#db_name#" \
-e "s#@DB_USER@#db_user#" \
-e "s#@DB_PASS@#db_pass#" mysql.sql > kisti.sql
```

3. Add the following line at the file created above if you want to access the database remotely

```
GRANT ALL ON db_name.* TO 'db_user'@'host_name' IDENTIFIED BY 'db_pass';
```

4. Build the database by using the SQL file

```
mysql -u root -h localhost -p < kisti.sql
```

Note that NFS (Network File System) needs to be set between the webserver and Tupelo repository server in order to provide web applications access to the files (or blobs) stored in File (Blob) Storage server. The user guide to set up NFS on the Ubuntu Linux platform is found at <https://help.ubuntu.com/community/SettingUpNFSHowTo>.

4.2 Semantic Content Repository Server

The semantic content repository server is a RESTful service that provides Tupelo context(s). The Context abstraction is the basis of Tupelo's flexible, distributed architecture for managing information. Each Context represents one or more sources of and/or destinations for information, including binary data, text data, and RDF metadata. Contexts act as brokers between an application and any of a variety of different ways of accessing, storing, querying, and managing information, including file systems, relational databases, web services, and RDF triple stores. In this project, the RESTful services are used by Cyberintegrator Engine. The following steps provide the instructions for setting up the Semantic Content Repository Server: (Note that the commands below are tested on the Linux platform)

1. (Optional) Create user and switch to this user (example: cyberintegrator)
2. Download the correct binary at
<https://opensource.ncsa.illinois.edu/jenkins/job/Repository%20Server%20Headless/>
3. Unzip the binary
4. Place the file, context.xml
 - a. Acquire an example context.xml file

```
wget --no-check-certificate -q -O context.xml \
https://opensource.ncsa.illinois.edu/confluence/download/attachments/589837/context.xml
```

- b. Replace database name (db_name), user name (db_user), user password (db_pass) for the repository database, and the location of data storage, then create a new context file (kisti.xml)

```
-----  
sed -e "s#@DB_SCHEMA@#db_name#" \  
-e "s#@DB_USER@#db_user#" \  
-e "s#@DB_PASS@#db_pass#" \  
-e "s#@FOLDER@#data_storage_location#" context.xml > kisti.xml  
-----
```

- c. Copy the new context file to ~/NCSA/ContextServer/context.xml

```
-----  
cp kisti.xml ~/NCSA/ContextServer/context.xml  
-----
```

- 5. Install the startup script
 - a. Copy the Cyberintegrator Server script to repository-server
 - b. Modify the script according to your setting
 - c. Make file executable

```
-----  
sudo chmod 755 /etc/init.d/repository-server  
-----
```

- d. Update rc and start the server for Ubuntu Linux

```
-----  
sudo update-rc.d repository-server defaults  
sudo /etc/init.d/repository-server start  
-----
```

The following URL gives you the status of the server: [http://\[hostname\]:9999/status](http://[hostname]:9999/status).

4.3 Cyberintegrator Engine Server

The Cyberintegrator (CI) Engine executes the CI workflow and monitors the status of the execution. The CI Engine server provides RESTful service. The following steps are setting up CI Engine Server: (Note that the commands below are tested on Linux platform)

1. (Optional) Create user and switch to this user (example: cyberintegrator)
2. Download the correct binary at
<https://opensource.ncsa.illinois.edu/jenkins/job/CyberintegratorServer/>
3. Unzip the binary
4. Install the startup script
 - a. Copy the example script to the location of startup script (for Ubuntu Linux)

```
-----  
cp [location of binary]/service/cyberintegrator.ubuntu /etc/init.d/cyberintegrator  
-----
```

- b. Modify the script according to your settings

- c. Make file executable

```
chmod 755 /etc/init.d/cyberintegrator
```

- d. Update rc and start the server (for Ubuntu Linux)

```
update-rc.d cyberintegrator defaults  
/etc/init.d/cyberintegrator start
```

4.4 Gondola Server

Currently, the Gondola binary is not available for download. Instead, you can check out the source code at <https://opensource.ncsa.illinois.edu/svn/gondola> and build the binary using the following steps:

1. Check out the project named, “gondola.team”
2. Import the team project set named, “gondola_project.psf”
3. Open “GondolaServer.product” under the project, “edu.illinois.ncsa.gondola.rest”
4. Click on “Eclipse Product export wizard” link under “Exporting” section. That wizard will step the user through the build process.

After you obtain the binary, please follow Step 4 in previous section to install the Gondola Server. When the Gondola service started, the service configuration file is required. The default one is shipped with the binary package as shown below:

Gondola_servic_cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<service-config xmlns="http://edu.illinois.ncsa.gondola/service"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="service_configuration.xsd">  
    <worker-pool>80</worker-pool>  
    <initialize-queue>20</initialize-queue>  
    <submit-queue>20</submit-queue>  
    <update-queue>20</update-queue>  
    <cancel-queue>20</cancel-queue>  
    <queue-refresh-in-secs>60</queue-refresh-in-secs>  
    <status-refresh-in-secs>30</status-refresh-in-secs>  
    <jdbc-access type="h2-submission-engine-access">  
        <jdbc-connection user="user" password="user"  
driverClass="org.h2.Driver">jdbc:h2:engine_store</jdbc-connection>  
        <initialize always="true">  
            <execute>create table XMLB ( key integer not null generated always  
                as identity, uuid varchar(255), submitted bigint, xml clob, primary key  
(key) )  
            </execute>  
            <execute>create index I_0000 on XMLB (uuid)</execute>  
            <execute>create index I_0001 on XMLB (submitted)</execute>
```

```

<execute>create table USER ( name varchar(63) not null, updated bigint, cert
blob, primary key (name) )
</execute>
<execute>create index I_0011 on USER (updated)</execute>
<execute>create table HANDLER ( uri varchar(255) not null, cmd varchar(255),
args varchar(255), parser blob, primary key (uri) )
</execute>
<execute>create table SCRIPT ( sid integer not null, script blob )</execute>
<execute>create index I_0020 on SCRIPT (sid)</execute>
<execute>alter table SCRIPT add FOREIGN KEY (sid) references XMLB (key)
</execute>
<execute>create table STATUS ( sid integer not null, job_id varchar(127),
work_dir varchar(255), log_file varchar(255), state varchar(31), ordinal
smallint, updated bigint, error blob, cleaned smallint, pinned smallint )
</execute>
<execute>create index I_0030 on STATUS (sid)</execute>
<execute>create index I_0031 on STATUS (job_id)</execute>
<execute>create index I_0032 on STATUS (ordinal)</execute>
<execute>create index I_0033 on STATUS (updated)</execute>
<execute>create index I_0034 on STATUS (cleaned)</execute>
<execute>create index I_0035 on STATUS (pinned)</execute>
<execute>alter table STATUS add FOREIGN KEY (sid) references XMLB (key)
</execute>
<execute>create table SUBMISSION ( sid integer not null, user varchar(63),
t_user varchar(63), t_uri varchar(255), u_home varchar(255), tu_home
varchar(255), sub_p varchar(255), cncl_p varchar(255), parser blob,
ehandler blob )
</execute>
<execute>create index I_0040 on SUBMISSION (sid)</execute>
<execute>create index I_0041 on SUBMISSION (user)</execute>
<execute>create index I_0042 on SUBMISSION (t_uri)</execute>
<execute>alter table SUBMISSION add FOREIGN KEY (sid) references XMLB (key)
</execute>
</initialize>
<drop onExit="true" />
<max-batch-insert>128</max-batch-insert>
<supports-relational-constraints>false</supports-relational-constraints>
</jdbc-access>
<myproxy-settings>
  <server-uri>[my_proxy_server]</server-uri>
  <proxy-refresh-in-mins>60</proxy-refresh-in-mins>
  <proxy-margin-in-mins>60</proxy-margin-in-mins>
</myproxy-settings>
</service-config>

```

Note that current Gondola implementation is using H2 database as shown in the XML above.

4.5 eAIRS Web Application

The eAIRS web is a GWT web application. It consists of two main parts: 1) javascripts for client side, and 2) servlet for server side. The deployed form of eAIRS is in WAR format. The WAR

file can be deployed to Java Servlet Containers such as Apache Tomcat or Jetty. In this document, Apache Tomcat 6.x is used to serve eAIRS web application.

4.5.1 Configuring Tomcat

Depending on your default setting and system setting, you may need to add the permission to the certain file access needed by eAIRS app. The MyProxy GWT RPC service requires using “/dev/urandom.” The following line needs to be added to the file CATALINA_HOME/conf/policy.d/04webapps.policy

CATALINA_HOME/conf/policy.d/04webapps.policy

```
grant {  
    .  
    .  
    permission java.io.FilePermission "/dev/urandom", "read";  
};
```

4.5.2 Deploying WAR

Deployment of WAR file at Tomcat is simple. You can copy the WAR file to the CATALINA_HOME/webapps folder or you can also use Tomcat Manager web application.

4.5.3 Configuring eAIRS

Three parameters for eAIRS are stored in web.xml; 1) URL for executing CI workflows, 2) URL for executing CI steps, and 3) the location of Apache Lucene index file. The below shows the example of the setting:

web.xml

```
.  
. .  
<web-app>  
    <context-param>  
        <param-name>execute.step.url</param-name>  
        <param-value>http://[CI_server]:8183/cyberintegrator/engine/steps/execute</param-value>  
    </context-param>  
    <context-param>  
        <param-name>execute.workflow.url</param-name>  
        <param-value> http://[CI_server]:8183/cyberintegrator/engine/workflows/execute </param-  
value>  
    </context-param>  
    <context-param>  
        <param-name>lucene.location</param-name>  
        <param-value>[absolute_path_of_folder_location]</param-value>  
    </context-param>  
. .  
</web-app>
```

Also, you need to update the file EAIRS/WEB_INF/classes/context.xml. The file contains the connection information to Tupelo Repository. Please refer the example context.xml file in Appendix B.

5 User Guide

5.1 Gondola

As explained previous sections, Gondola is the lightweight and flexible HPC job submission and monitoring system that provides RESTful services. It provides the following REST API:

Table 1. Gondola REST API

Request Method	URL Pattern	Description
POST	/jobs/submit	Submit job submission xml file and return the status with job id.
GET	/jobs/{job_id}	Get the status of the job with {job_id}
PUT	/jobs/{job_id}/cancel	Cancel the job with {job_id}
PUT	/jobs/{job_id}/delete	Delete the entry with {job_id} from the database

In the current implementation, CI tool is submitting jobs via Gondola REST API described above. The job is described in the job submission XML file. Note that the XML XSDs for the service definition and job submission supported by Gondola are provided in Appendix A.

The code below is an example job submission XML file:

```

Job_submission.xml

<?xml version="1.0" encoding="UTF-8"?>
<job-submission xmlns="http://edu.illinois.ncsa.gondola/submission"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="job_submission.xsd" id="test-submission">
  <myproxy-user>[username]</myproxy-user>
  <!-- Cyber user -->
  <target-user>[username]</target-user>
  <!-- Cyber gsissh access string -->
  <target-uri>gsissh://[hostname]</target-uri>
  <service-user-home>[user-home-directory-client-side]</service-user-home>
  <!-- Home directory at Cyber -->
  <target-user-home>[user-home-directory-HPC-side]</target-user-home>
  <submit-path>/opt/ibmll/LoadL/full/bin/llsubmit -X kisti.baekdusan</submit-path>
  <terminate-path>/opt/ibmll/LoadL/full/bin/llcancel -X kisti.baekdusan</terminate-path>
  <submit-error-handler>
    <stderr-warn flags="DOTALL / UNIX_LINES">.*will be charged to:.*</stderr-warn>
    <stderr-error flags="DOTALL / UNIX_LINES">.*not been submitted.*</stderr-error>
  </submit-error-handler>
  <job-id-parser>
    <!-- llsubmit: The job "cyber.kisti.re.kr.562" has been submitted. -->
    <!-- need to extract 562 -->
  </job-id-parser>
</job-submission>
```

```

<expression>.*cyber.kisti.re.kr.([\d]+).*</expression>
<job-id-group>1</job-id-group>
</job-id-parser>
<status-handler>
  <command-path>/opt/ibm11/LoadL/full/bin/l1q -X kisti.baekdusan</command-path>
  <parser>
<expression>.+[.][\d]+[.][\d]+[\s]+[\S]+[\s]+[\S]+[\s]+([\S]+)[\s]+.*<expression>
</status-handler>
<script deleteAfterSubmit="false">
<line>#!/bin/bash</line>
<line># @ job_name = submit_test</line>
<line># @ job_type = serial</line>
<line># @ step_name = pgaia_mpi</line>
<line># @ class = normal</line>
<line># @ notify_user = [user_email]</line>
<line># @ notification = complete</line>
<line># @ wall_clock_limit=2:00:00</line>
<line># @ resources=ConsumableCpus(1) ConsumableMemory(1gb)</line>
<line># @ wall_clock_limit = 2:00:00</line>
<line># @ output = $(job_name).$(step_name).out</line>
<line># @ error = $(job_name).$(step_name).err</line>
<line># @ queue</line>
<line>export MP_SHARED_MEMORY=yes</line>
<line>export MP_SINGLE_THREAD=yes</line>
<line>TS=$( date +\%s )</line>
<line>SCRWD="$SCR/$$-$TS.tmp"</line>
<line>mkdir $SCRWD</line>
<line>cd $SCRWD</line>
<line log="true">pwd</line>
<line>sleep 300</line>
<line>/bin/uname -a &gt; uname.out</line>
<line log="true">echo DONE</line>
</script>
<x509-certificate>
[usre_X509_certificate]
</x509-certificate>
</job-submission>

```

5.2 How to Create and Export a New Workflow

In this section, we are going to work through the steps of creating a new workflow and then demonstrate how a user can export a new workflow and import it into a different repository so it can be published and ran through the web UI. This example will use information for my account and the Ranger HPC system, but all parameters should be applicable for a typical HPC system.

- First, go to File > New > Workflow. This should bring up a wizard that allows you to provide a title for the new workflow, see Figure 7. In this example, we will name our new workflow “My eAIRS Simulation”. Click “Finish” after specifying a title. You should see a blank workflow graph in the “Workflow Graph” view.

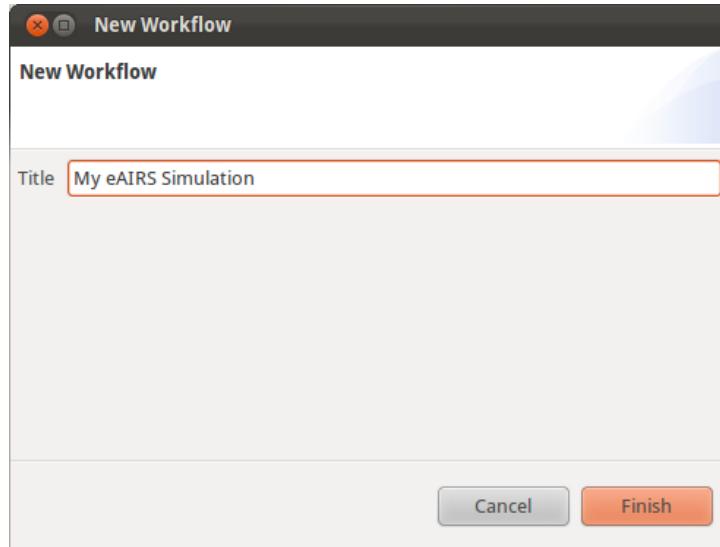


Figure 7. New Workflow Dialog

Next, we need to add some tools to our workflow. The first tool we are going to add is called “eAIRS Parameter Tool”. This tool will allow users to generate a file containing the input parameters they want to use in the eAIRS CFD Solver. To add it to the workflow graph, select “eAIRS Parameter Tool (v1.0)” in the “Tools” view and click the “Create Step” button. See Figure 8 for what my tools view looks like, yours should be similar.

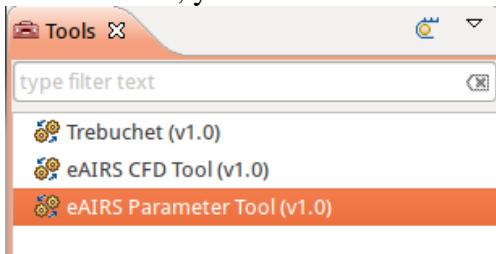


Figure 8. Tools View

After clicking the “Create Step” button, you should see a “Tool Execution” view similar to Figure 9 that will allow you to specify the parameters before executing the tool.

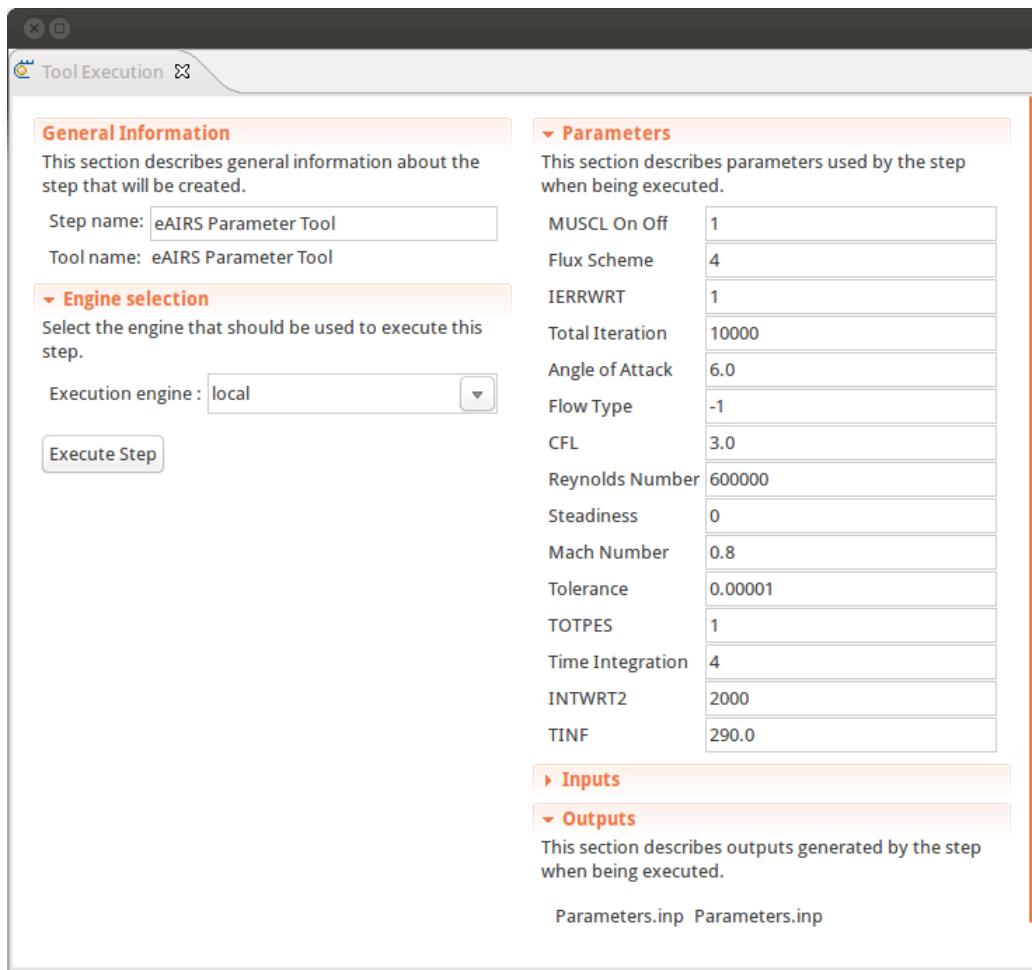


Figure 9. Tools Execution View for eAIRS Parameter Tool

For this example, the default parameters are sufficient and you can click on the “Execute Step” button. After executing that step, your workflow graph should look like Figure 10. You will see the name of the tool and the outputs that it generated (in this case, a parameters file).



Figure 10. Workflow Graph with eAIRS Parameter Tool

You should also have a file in the data view called “Parameters.inp”.

The next tool we need to add to our workflow is the Trebuchet (v1.0) tool. This tool uses GridFTP to transfer data from the repository to the HPC machine for execution. To add this tool to our workflow, select “Trebuchet (v1.0)” in the “Tools” view and click the “Create Step” button. This should open the tool in the “Tool Execution” view and allow you to edit the input parameters. For this example, enter the following information in the “Parameters” section, replacing my values with your information:

- Where it says “Remote Path”, enter the path to your home directory:
“/share/home/01438/cnavarro”

- Where it says “Target GridFTP”, enter the gridftp hostname and port:
“gridftp://gridftp.ranger.tacc.teragrid.org:2811”
- Where it says “CA Certificates”, enter the path for the host certificates:
“/home/cnavarro/workspaces/knsg/ncsa.ca.certs/certificates”

In the “Inputs” section, enter the following information:

- For “Input Parameters”, it should already have picked up the file called “Parameters.inp” we generated in the previous step. If it hasn't, select the file in the “Data” view and drag and drop it onto the triangular button of the dropdown box for that field.
- For “Mesh”, select the file “eAIRS_naca15.msh” from the “Data” view and drag and drop the file onto the triangular button of the dropdown box for that field. You should now see that file specified for the mesh input.

Your “Tool Execution” view should now look similar to the one in Figure 11.

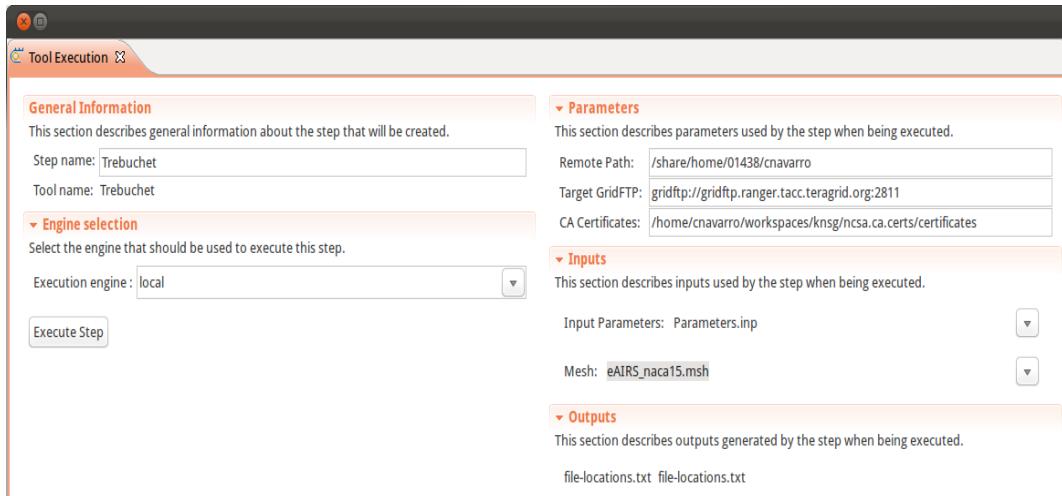


Figure 11. Tool Execution View for Trebuchet Tool

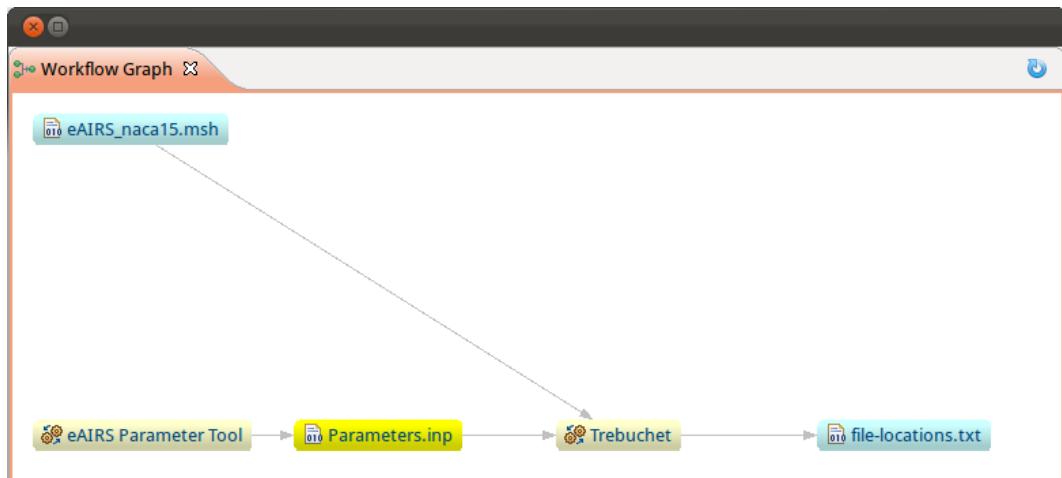


Figure 12. Workflow Graph with eAIRS Parameter Tool and Trebuchet

All other parameters can be left as the default. Click the “Execute Step” button to execute this tool. This will execute the tool and add it to your workflow graph. If you go to the “Workflow Graph” view, you should see something similar to Figure 12.

The last tool we need to add to our workflow is the eAIRS CFD Tool. This tool will execute the eAIRS 2D Solver on the HPC machine we specify and retrieve the results when the job is finished. To add this tool to the workflow, select “eAIRS CFD Tool (v1.0)” and click the “Create Step” button. This should open the tool in the “Tool Execution” view. For this example, enter the information below in the “Parameters” section:

- Where it says “Target Username”, enter your username for the HPC machine you will be executing the tool on: “cnavarro”
- Where it says “MyProxy Username”, enter your MyProxy username: “cnavarro”
- Where it says “Rest Endpoint”, enter the rest endpoint of the REST service that will launch the job: “<http://kisti.ncsa.illinois.edu:8081/rest>”
- Where it says “Target Userhome”, enter the path to your home directory on the HPC machine: “/share/home/01438/cnavarro”
- Where it says “CA Certificates”, enter the path for the host certificates: “/home/cnavarro/workspaces/knsg/ncsa.ca.certs/certificates”
- Where it says “Target GridFTP”, enter the gridftp hostname and port: “gridftp://gridftp.ranger.tacc.teragrid.org:2811”
- Where it says “HPC Target”, specify the gsishh hostname to launch the job on: gsishh://tg-login.ranger.tacc.teragrid.org

In the “Inputs” section, we need to specify the workflow script and the locations of the files on the HPC machine that will be used in the script. The locations of the files on the HPC machine were generated by the previous tool “Trebuchet” (file-locations.txt) and will be the input for that field.

- Where it says “Workflow”, drag and drop the workflow script file that you want to run onto the triangular button of the dropdown box for that field, in my case it is: “eAIRS-Ranger-Workflow.xml”
- Where it says “File Locations”, the dropdown box should contain the “file-locations.txt” file that was generated by the previous tool. Select it if it isn’t already selected. If it doesn’t contain that file, find the “file-locations.txt” file in the “Data” view and drag and drop the file onto the triangular button of the dropdown box to add it.

Your “Tool Execution” view should now look similar to the one in Figure 13.

All other parameters can be left as the default. The “Outputs” section is specified by the tool and the user doesn’t need to adjust those. To execute the tool, click the “Execute Step” button. This will execute the tool and add it to your workflow graph. If you go to the “Workflow Graph” view, you should see something similar to Figure 14.

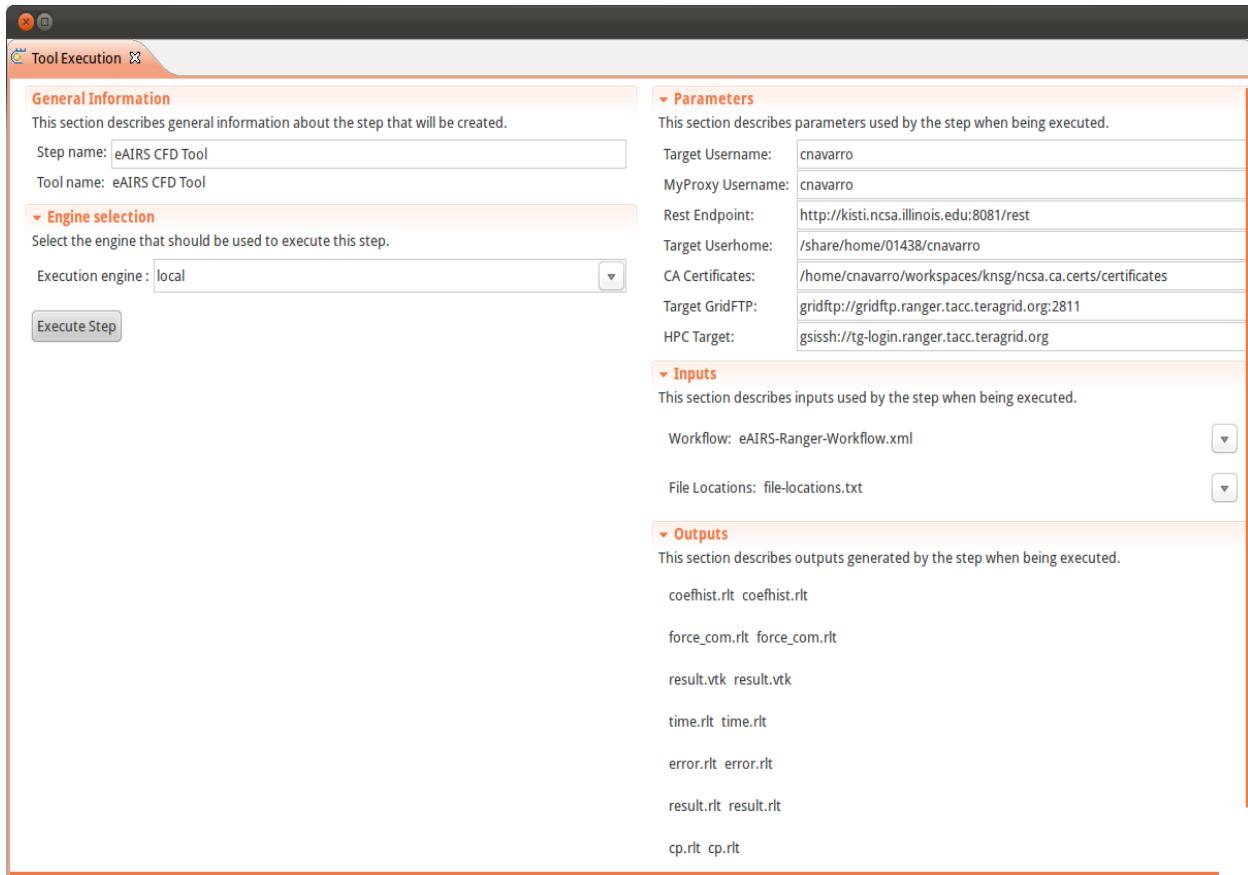


Figure 13. Tool Execution View for eAIRS CFD Tool

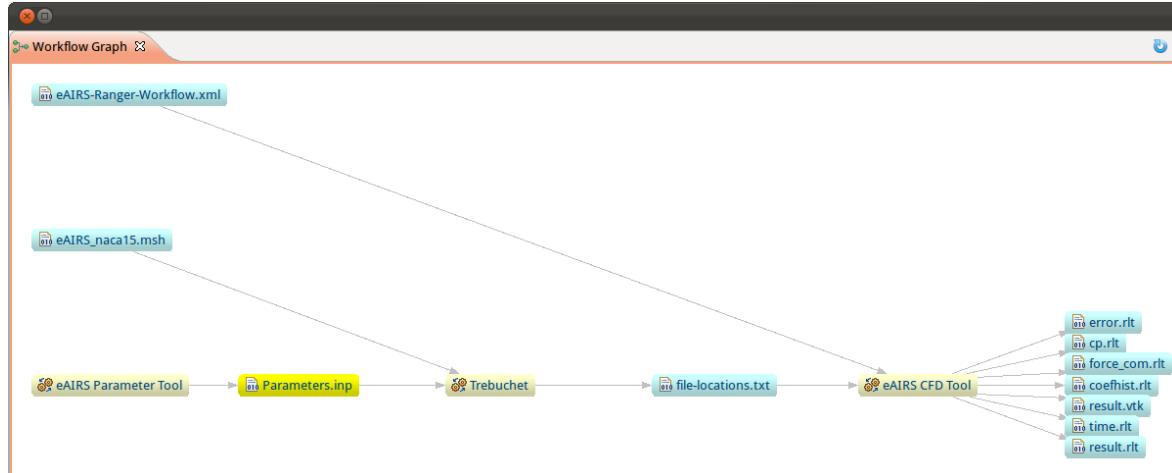


Figure 14. Complete Workflow Graph

Now that we have a complete workflow, we need to export this workflow and import it into the repository that our web application will talk to so you can publish and execute workflows from the web. The section below will describe how you can export the workflow and tools if the repository containing your new workflow is different than the one the web application talks to. To export the workflow, we need to change Cyberintegrator's perspective to the "Workflow Execution" perspective. To do this, go to Window > Open Perspective > Workflow Execution.

There you should see a list of your workflows, including the newly created “My eAIRS Simulation”. To export this workflow, do the following:

- In the “Workflows” view, select “My eAIRS Simulation”. This should display the information about the workflow. See Figure 15.

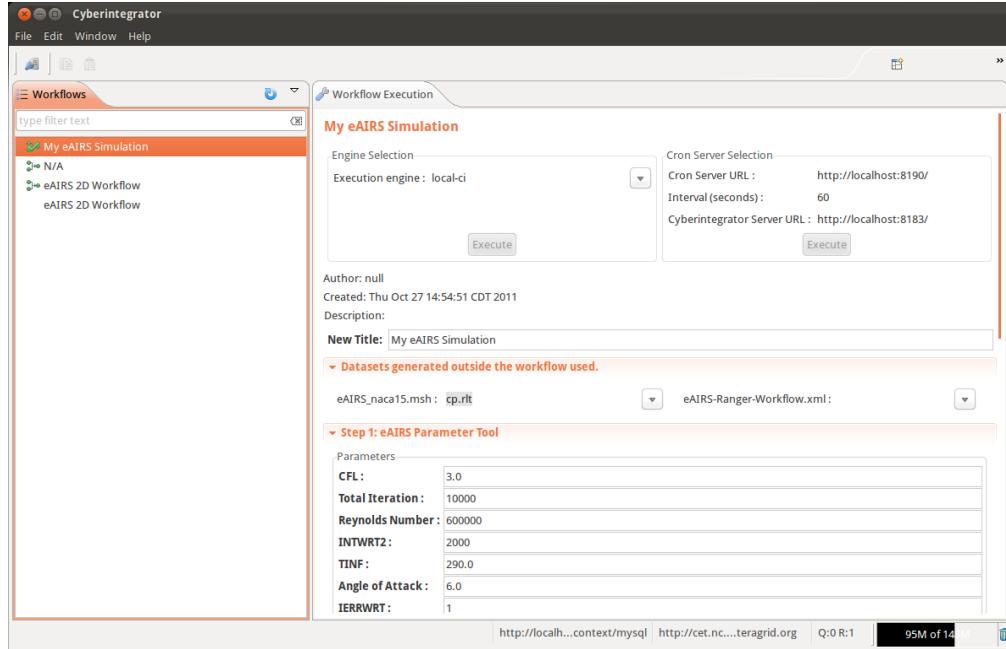


Figure 15. Workflow Information

- After selecting the workflow, go to File > Export. This should bring up a dialog similar to the one in Figure 16.

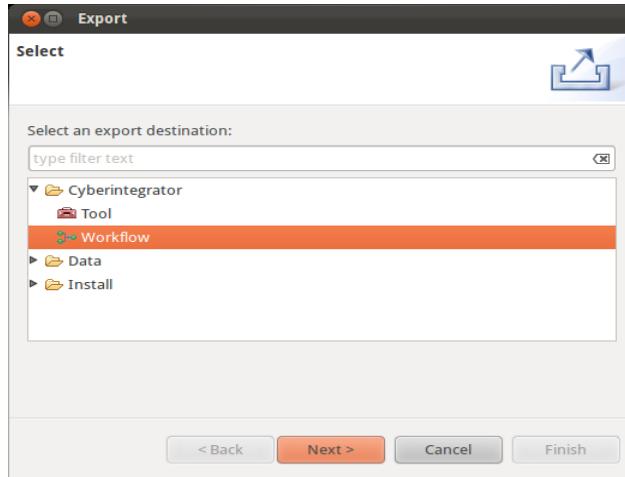


Figure 16. Export Workflow Dialog

- Go to the “Cyberintegrator” category and select “Workflow”. Click Next.
- Use the “...” browse button to select a location to export the workflow and a name for the exported file or just enter the location and file name (e.g.

`/home/cnavarro/eAIRSWorkflow.zip`). If you want to include the data created and used by this workflow (e.g. `eAIRS_naca15.msh`), check the box to export it with the workflow. However, you can leave this box unchecked if you are publishing this workflow for the web application since you can expose all of the dataset fields so users can specify the inputs. Click “Finish” to export the workflow. Note: if you are including all data and want to include the result files as part of the export, you will need to wait for the workflow to finish or the result datasets will be empty. For this demonstration, I am going to include all data in my export. See Figure 17.

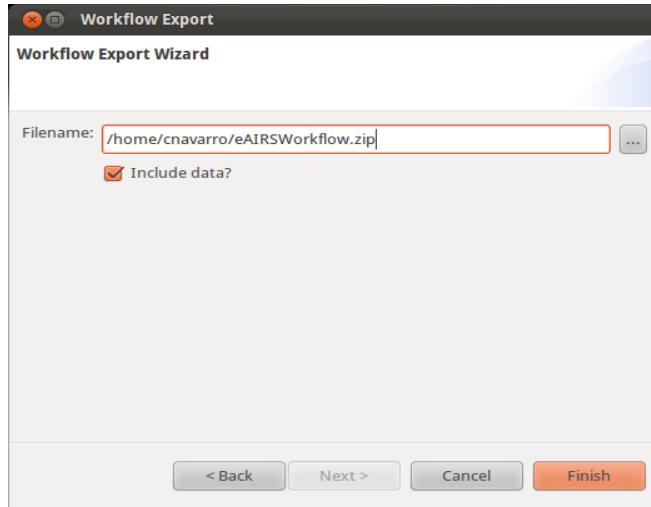


Figure 17. Specify Name and Location of Exported Workflow

Now that we have exported our workflow, we can connect to another repository with Cyberintegrator and import our workflow. Connect to another repository (e.g. the web application’s repository) with Cyberintegrator and do the following:

- Go to File > Import. You should see a dialog similar to the one in Figure 18.

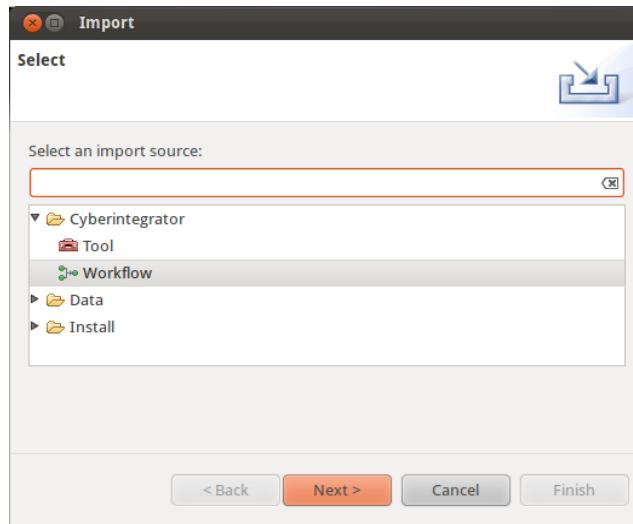


Figure 18. Import Dialog

- Go to the “Cyberintegrator” category and select “Workflow”. Click Next.
- Use the “...” browse button to find the file we just exported or enter the path to the file in the textfield. Your dialog page should look similar to the one in Figure 19. Click Finish to import the workflow.

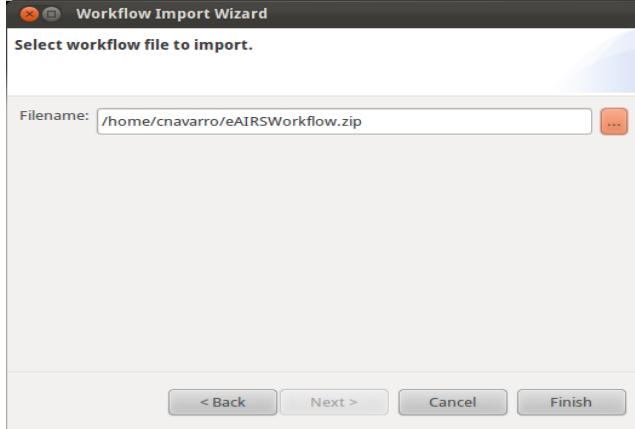


Figure 19. Workflow to Import into Repository

The workflow should now be visible in the “Workflow Execution” view and available to the web application for publishing.

5.3 eAIRS Web Application

The user needs to login eAIRS web application. The user is also able to change the MyProxy server in the login page as below.

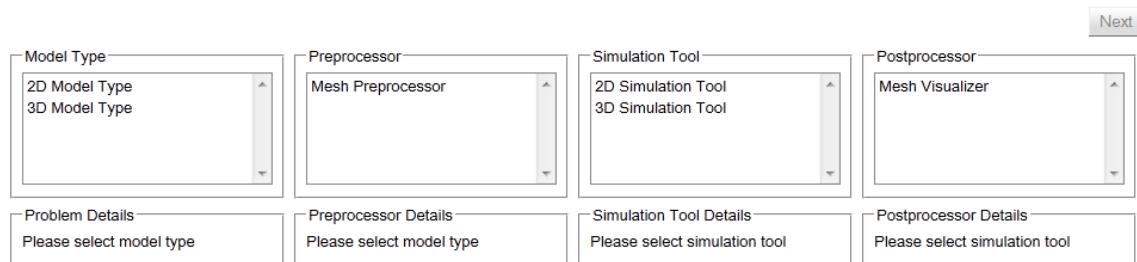
Sign in to your account	
Username:	<input type="text"/>
Password:	<input type="password"/>
MyProxy Server URI	<input type="text" value="ssh://myproxy.teragrid.org:7512"/>
<input type="button" value="Login"/> <input type="button" value="Offline"/>	
Please login...	

Figure 20. Login Page

After the login, the user will redirect to “Create” simulation page. The user can select a published workflow by choosing the preprocessor, simulation tool, and postprocessor as shown in Figure 21.

After the user clicked “Next” button, the “Launch” page will show for the user to enter the input parameter with the widget mapped to the workflow as shown in Figure 22. Note that there are three kinds of widget to select a mesh file: 1) select from the existing mesh file in the repository, 2) upload the mesh file, and 3) mesh applet.

Create Simulation

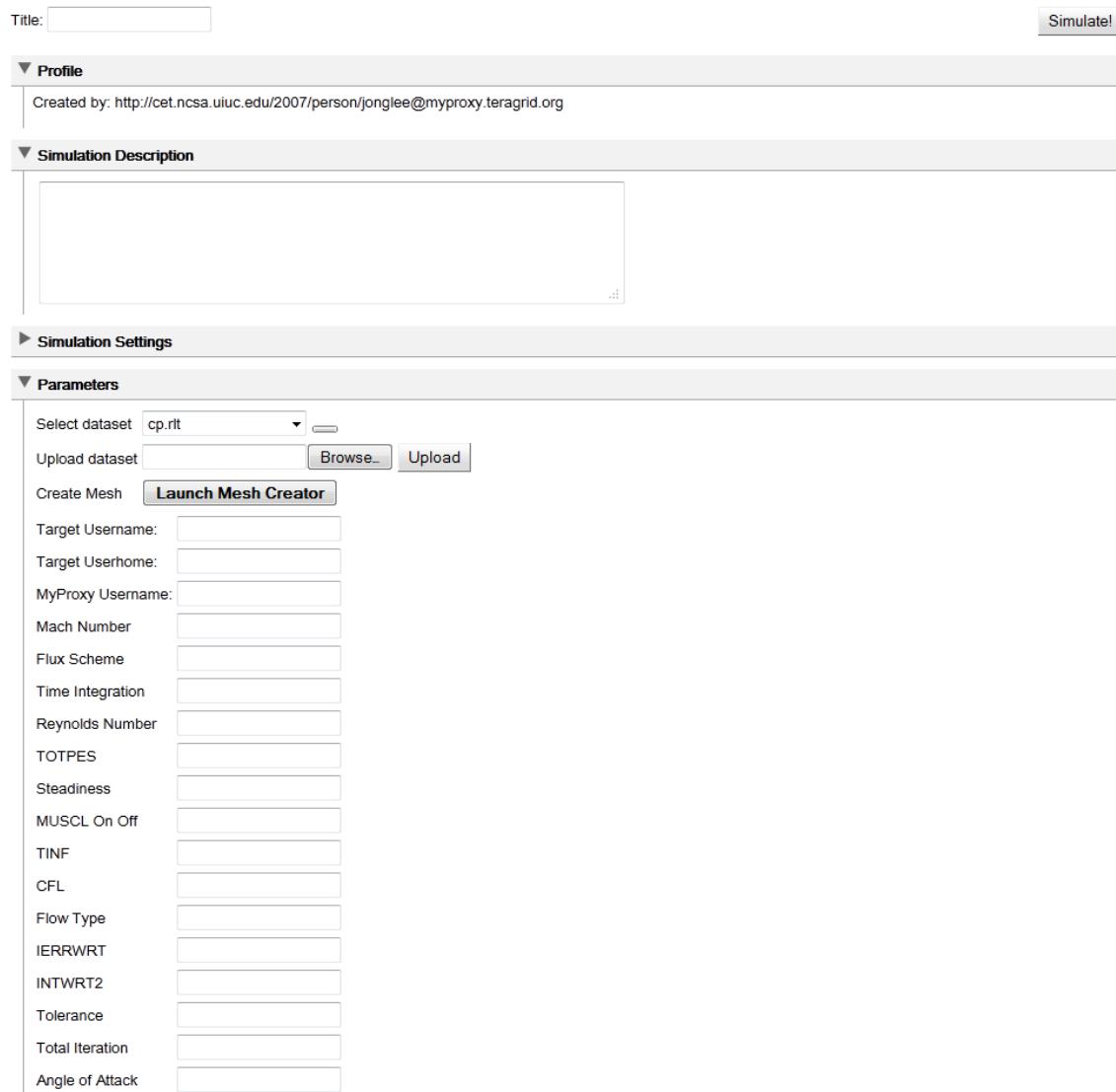


The screenshot shows the 'Create Simulation' page. It consists of four main sections: 'Model Type', 'Preprocessor', 'Simulation Tool', and 'Postprocessor'. Each section has a dropdown menu with options and a 'Details' box below it. A 'Next' button is located in the top right corner.

Model Type	Preprocessor	Simulation Tool	Postprocessor
2D Model Type 3D Model Type	Mesh Preprocessor	2D Simulation Tool 3D Simulation Tool	Mesh Visualizer
Problem Details Please select model type	Preprocessor Details Please select model type	Simulation Tool Details Please select simulation tool	Postprocessor Details Please select simulation tool

Figure 21. Create a Simulation Page

Launch New Simulation



The screenshot shows the 'Launch New Simulation' page. It includes fields for 'Title' and a 'Simulate!' button. Below these are sections for 'Profile' (with a note about creation by a specific user), 'Simulation Description' (with a large text area), 'Simulation Settings' (with a note), and 'Parameters'. The 'Parameters' section contains numerous input fields for simulation parameters like dataset selection, mesh creation, and various numerical settings.

Profile
Created by: http://cet.ncsa.uiuc.edu/2007/person/jonglee@myproxy.teragrid.org

Simulation Description
[Large Text Area]

Simulation Settings
[Note: ...]

Parameters
Select dataset: cp.rlt
Upload dataset: <input type="text"/> <input type="button" value="Browse..."/> <input type="button" value="Upload"/>
Create Mesh: <input type="button" value="Launch Mesh Creator"/>
Target Username: <input type="text"/>
Target Userhome: <input type="text"/>
MyProxy Username: <input type="text"/>
Mach Number: <input type="text"/>
Flux Scheme: <input type="text"/>
Time Integration: <input type="text"/>
Reynolds Number: <input type="text"/>
TOTPES: <input type="text"/>
Steadiness: <input type="text"/>
MUSCL On Off: <input type="text"/>
TINF: <input type="text"/>
CFL: <input type="text"/>
Flow Type: <input type="text"/>
IERRWRT: <input type="text"/>
INTWRT2: <input type="text"/>
Tolerance: <input type="text"/>
Total Iteration: <input type="text"/>
Angle of Attack: <input type="text"/>

Figure 22. Launch Simulation Page

The Figure 23 shows the mesh generator (or viewer) applet in eAIRS web application.

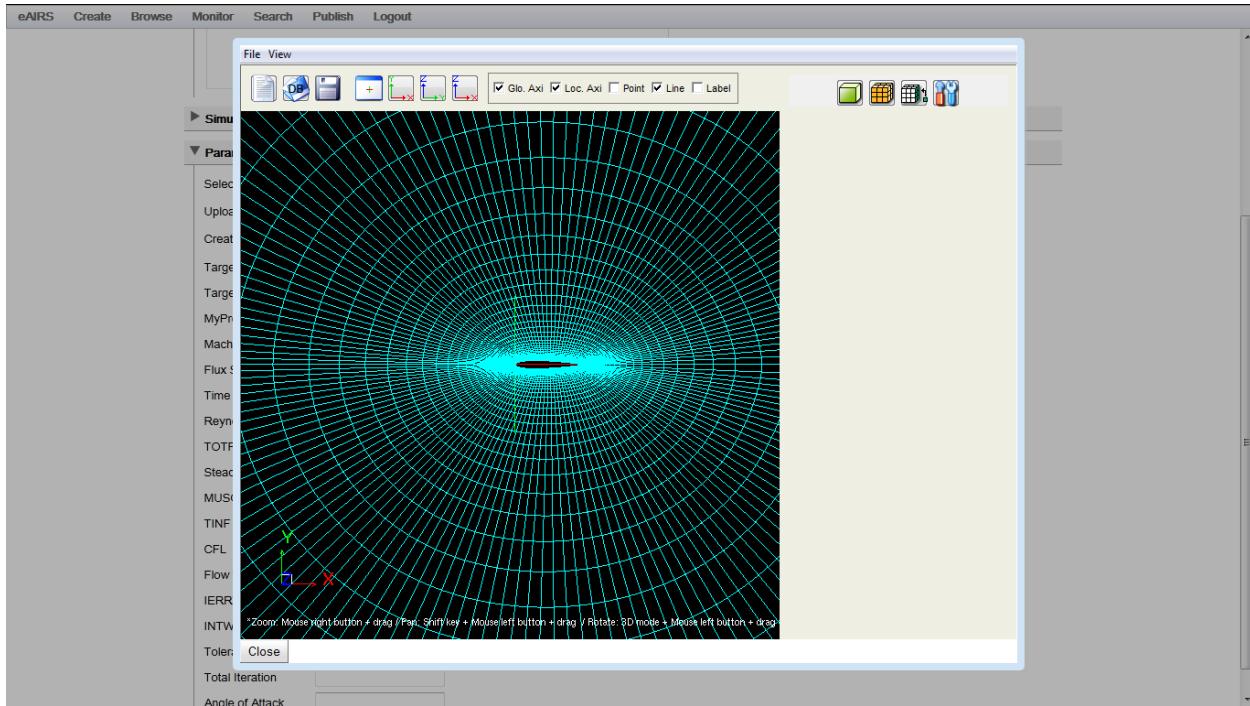


Figure 23. Mesh Generator (Viewer)

After the user finished entering the values for all parameters, the user can launch the workflow by clicking “Simulate” button. After the launch, the user will be redirected to “Monitor and Browse” simulations page as shown in Figure 24.

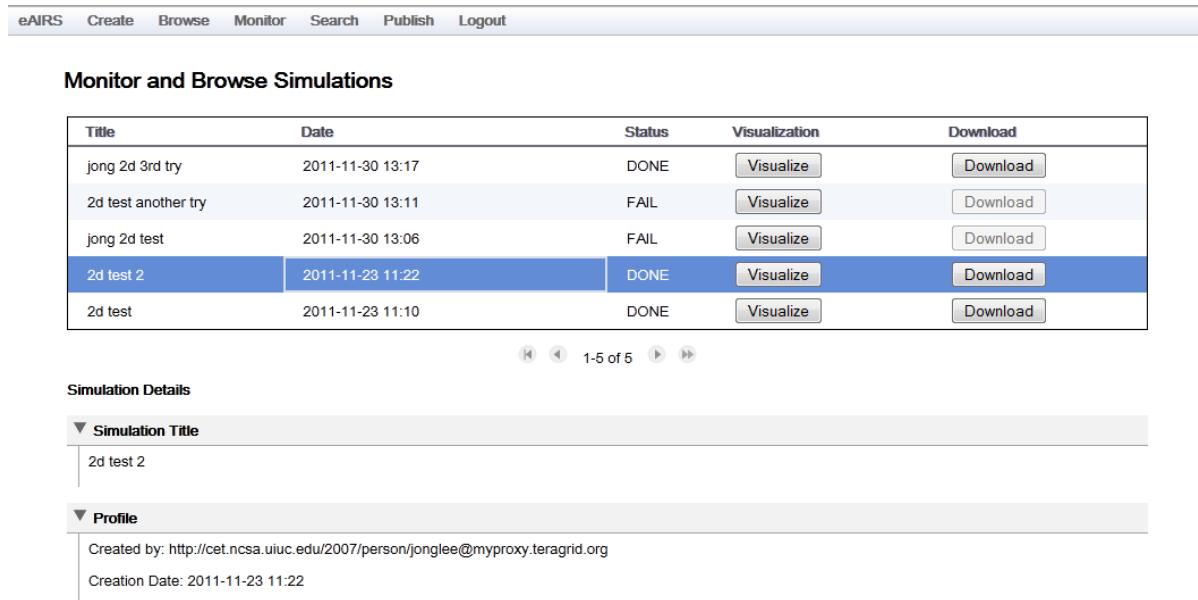


Figure 24. Monitoring and Browsing Page

The user also can search their simulations at “Search” page as shown in Figure 25.

eAIRS Create Browse Monitor Search Publish Logout

Search Simulations

2d

Display: My Simulations All Simulations

Title	Date	Status	Visualization	Download
jong 2d test	2011-11-30 13:06	FAIL	<input type="button" value="Visualize"/>	<input type="button" value="Download"/>
2d test another try	2011-11-30 13:11	FAIL	<input type="button" value="Visualize"/>	<input type="button" value="Download"/>
jong 2d 3rd try	2011-11-30 13:17	DONE	<input type="button" value="Visualize"/>	<input type="button" value="Download"/>

◀ ▶ 1-3 of 3 ▶▶

Figure 25. Search Page

6 Future work

The following are future work that has been identified:

- 1) Developing web application for creating workflows through a web interface, similar to what CI editor did.
- 2) Developing Science AppStore that manages the executable, source code, or workflow to simplify sharing and executing workflows.
- 3) Developing an extension of Eclipse PTP to submit the executable to the Science AppStore.
- 4) Developing a web application for different domain using KNSG-AF.
- 5) Optimizing and enhancing KNSG-AF
- 6) Improving Gondola
- 7) Develop a mobile extension to KNSG-AF to provide a UI more suitable to mobile devices that would allow users to submit and monitor jobs through their mobile device and/or utilize social media services such as twitter to get updates about their job(s)
- 8) Develop mobile version of eAIRS using KNSG-AF

These future works will enhance what we've developed thus far and developing a Science AppStore and related technology would enable scientists and researchers to easily create and deploy their applications for broader use in the academic community to facilitate scientific discovery.

7 Statement of Expenses

This is the statement of Expenses as of September 30, 2011.

- **Project Name:** Developing Semantically-aware and Web-enabled KNSG Application Framework
- **Project Period:** February 15, 2011 – December 15, 2011
- **Project Budget:** \$148,344
- **Project Investigator:** Danny Powell
- **Institute name:** The Board of Trustees of the University of Illinois
- **Budget-Expense Specification (USD)**

	Budget (A)	Expense (B)	Increase or Decrease (A-B)	Reason
1 Expenses of Direct Costs				
◦ Personal Salaries and wage	\$63,496.00	\$41,217.41	\$22,278.59	
◦ Fringe Benefits	\$22,597.00	\$16,203.61	\$6,393.39	
◦ Expense of Other Direct Cost	\$7,500.00	\$3,044.13	\$4,455.87	
- Travel	\$5,000.00	\$1,788.04	\$3,211.96	
- Material Fees	\$0.00	\$0.00	\$0.00	
- Others	\$2,500.00	\$1,256.09	\$1,243.91	
2 Expenses of Indirect Costs	\$54,751.00	\$35,372.13	\$19,378.87	
3 Total Expense(1+2)	\$148,344.00	\$95,837.28	\$52,506.72	

Appendix A: XSD Schema definition for Gondola

Job_submission.xsd: XSD for job submission

```
<?xml version="1.0" encoding="UTF-8"?>
<xss:schema targetNamespace="http://edu.illinois.ncsa.gondola/submission"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:subm="http://edu.illinois.ncsa.gondola/submission"
  elementFormDefault="qualified">
  <xss:element name="job-submission" type="subm:job-submission-type" />
  <xss:element name="job-status-parser" type="subm:parser-type" />
  <xss:element name="job-error-handler" type="subm:error-handler-type" />
  <xss:element name="job-script" type="subm:script-type" />
  <xss:element name="job-status-list" type="subm:job-status-list-type" />
  <xss:element name="job-status" type="subm:job-status-type" />
  <xss:element name="id-list" type="subm:id-list-type" />
  <xss:complexType name="job-status-list-type">
    <xss:sequence>
      <xss:element name="job-status" minOccurs="0" maxOccurs="unbounded"
        type="subm:job-status-type" />
    </xss:sequence>
  </xss:complexType>
  <xss:complexType name="id-list-type">
    <xss:sequence>
      <xss:element name="id" minOccurs="0" maxOccurs="unbounded" type="xs:string" />
    </xss:sequence>
  </xss:complexType>
  <xss:complexType name="job-submission-type">
    <xss:sequence>
      <xss:element minOccurs="0" name="instance-id" type="xs:string">
        <xss:annotation>
          <xss:documentation>service is agnostic of the structure of this id; it
just needs to be globally unique</xss:documentation>
        </xss:annotation>
      </xss:element>
      <xss:element minOccurs="0" name="myproxy-user" type="xs:string" />
      <xss:element minOccurs="0" name="target-user" type="xs:string" />
      <xss:element minOccurs="0" name="target-uri" type="xs:string" />
      <xss:element minOccurs="0" name="service-user-home" type="xs:string" />
      <xss:element minOccurs="0" name="target-user-home" type="xs:string" />
      <xss:element minOccurs="0" name="submit-path" type="xs:string" />
      <xss:element minOccurs="0" name="terminate-path" type="xs:string" />
      <xss:element minOccurs="0" name="submit-error-handler" type="subm:error-
handler-type" />
      <xss:element minOccurs="0" name="job-id-parser" type="subm:parser-type" />
      <xss:element minOccurs="0" name="status-handler" type="subm:status-handler-
type" />
      <xss:element minOccurs="0" name="script" type="subm:script-type" />
      <xss:element minOccurs="0" name="x509-certificate" type="xs:string" />
    </xss:sequence>
    <xss:attribute name="id" type="xs:string" />
  </xss:complexType>
```

```

<xs:complexType name="status-handler-type">
  <xs:sequence>
    <xs:element name="command-path" type="xs:string" />
    <xs:element name="command-args" type="xs:string" minOccurs="0" />
    <xs:element name="parser" type="subm:parser-type" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="error-handler-type">
  <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" name="stderr-warn"
type="subm:regex-type" />
    <xs:element minOccurs="0" maxOccurs="unbounded" name="stderr-error"
type="subm:regex-type" />
    <xs:element minOccurs="0" maxOccurs="unbounded" name="stdout-warn"
type="subm:regex-type" />
    <xs:element minOccurs="0" maxOccurs="unbounded" name="stdout-error"
type="subm:regex-type" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="parser-type">
  <xs:sequence>
    <xs:element name="expression" type="subm:regex-type"/>
    <xs:element name="job-id-group" type="xs:int" default="0" />
    <xs:element name="translator" type="subm:expression-group-translator-type"
minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="stderr" type="xs:boolean" default="false" />
</xs:complexType>
<xs:complexType name="regex-type" mixed="true">
  <xs:attribute name="flags" type="xs:string" />
  <xs:attribute name="split" type="xs:boolean" default="false" />
</xs:complexType>
<xs:complexType name="expression-group-translator-type">
  <xs:attribute name="group" type="xs:int" />
  <xs:attribute name="op" default="EQUALS">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="EQUALS" />
        <xs:enumeration value="EQUALS_IGNORE_CASE" />
        <xs:enumeration value="STARTS_WITH" />
        <xs:enumeration value="ENDS_WITH" />
        <xs:enumeration value="CONTAINS" />
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="from" type="xs:string" />
  <xs:attribute name="to" type="subm:job-state-type" />
</xs:complexType>
<xs:complexType name="script-type">
  <xs:sequence>
    <xs:element name="line" type="subm:line-type" minOccurs="0"
maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="deleteAfterSubmit" type="xs:boolean" default="true" />

```

```

</xs:complexType>
<xs:complexType mixed="true" name="line-type">
    <xs:attribute name="log" type="xs:boolean" default="false" />
</xs:complexType>
<xs:complexType name="job-status-type">
    <xs:sequence>
        <xs:element name="instance-id" minOccurs="0" type="xs:string" />
        <xs:element name="job-id" minOccurs="0" type="xs:string" />
        <xs:element name="working-directory" minOccurs="0" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="state" type="subm:job-state-type" />
</xs:complexType>
<xs:simpleType name="job-state-type">
    <xs:restriction base="xs:string">
        <xs:enumeration value="JOB_UNINITIALIZED" />
        <xs:enumeration value="JOB_INIT_PENDING" />
        <xs:enumeration value="JOB_INITIALIZED" />
        <xs:enumeration value="JOB_SUBMIT_PENDING" />
        <xs:enumeration value="JOB_SUBMITTED" />
        <xs:enumeration value="JOB_QUEUED" />
        <xs:enumeration value="JOB_RUNNING" />
        <xs:enumeration value="JOB_SUSPENDED" />
        <xs:enumeration value="JOB_CANCEL_REQUESTED" />
        <xs:enumeration value="JOB_CANCEL_PENDING" />
        <xs:enumeration value="JOB_CANCEL_FAILED" />
        <xs:enumeration value="JOB_CANCELED" />
        <xs:enumeration value="JOB_COMPLETED" />
        <xs:enumeration value="JOB_INIT_FAILED" />
        <xs:enumeration value="JOB_SUBMIT_FAILED" />
        <xs:enumeration value="JOB_SUCCESS" />
        <xs:enumeration value="JOB_FAILURE" />
    </xs:restriction>
</xs:simpleType>
</xs:schema>

```

Service_configuration.xsd: XSD for Gondola Service Configuration

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://edu.illinois.ncsa.gondola/service"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:service="http://edu.illinois.ncsa.gondola/service"
elementFormDefault="qualified">
    <xs:element name="service-config" type="service:service-type" />
    <xs:element name="jdbc-access" type="service:access-type" />
    <xs:complexType name="service-type">
        <xs:sequence>
            <xs:element name="worker-pool" type="xs:int" default="50" />
            <xs:element name="initialize-queue" type="xs:int" default="10" />
            <xs:element name="submit-queue" type="xs:int" default="10" />
            <xs:element name="update-queue" type="xs:int" default="10" />
            <xs:element name="cancel-queue" type="xs:int" default="10" />
        </xs:sequence>
    </xs:complexType>
</xs:schema>

```

```

<xs:element name="queue-refresh-in-secs" type="xs:int" default="60" />
<xs:element name="status-refresh-in-secs" type="xs:int" default="60" />
<xs:element name="jdbc-access" type="service:access-type" />
<xs:element name="myproxy-settings" type="service:my-proxy-type"
minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="access-type">
<xs:sequence>
<xs:element name="jdbc-connection" type="service:connection-type" />
<xs:element name="jdbc-dbcp" minOccurs="0" type="service:dbcp-type" />
<xs:element name="initialize" minOccurs="0" type="service:initialize-type" />
<xs:element name="drop" minOccurs="0" type="service:drop-type" />
<xs:element name="max-batch-insert" minOccurs="0" type="xs:int" default="128"
/>
<xs:element name="supports-relational-constraints" minOccurs="0"
type="xs:boolean" default="false" />
</xs:sequence>
<xs:attribute name="type" type="xs:string" />
</xs:complexType>
<xs:complexType name="connection-type" mixed="true">
<xs:attribute name="user" type="xs:string" />
<xs:attribute name="password" type="xs:string" />
<xs:attribute name="driverClass" type="xs:string" />
</xs:complexType>
<xs:complexType name="dbcp-type">
<xs:sequence>
<xs:element name="property" minOccurs="0" maxOccurs="unbounded">
<xs:complexType mixed="true">
<xs:attribute name="name" type="xs:string" />
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="name" type="xs:string" />
</xs:complexType>
<xs:complexType name="initialize-type">
<xs:sequence>
<xs:element name="execute" maxOccurs="unbounded" type="xs:string" />
</xs:sequence>
<xs:attribute name="always" type="xs:boolean" default="false" />
</xs:complexType>
<xs:complexType name="drop-type">
<xs:attribute name="onExit" type="xs:boolean" default="false" />
</xs:complexType>
<xs:complexType name="my-proxy-type">
<xs:sequence>
<xs:element name="server-uri" type="xs:string"/>
<xs:element name="host-cert-dir" type="xs:string" minOccurs="0"/>
<xs:element name="proxy-refresh-in-mins" type="xs:int" default="60"/>
<xs:element name="proxy-margin-in-mins" type="xs:int" default="120"/>
</xs:sequence>
</xs:complexType>
</xs:schema>

```

Appendix B: An example Context.xml

Context.xml: Tupelo Context

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:a="http://cet.ncsa.uiuc.edu/2007/context/"
  xmlns:b="tag:tupeloproject.org,2006:/2.0/beans/2.0/"
  xmlns:cet="http://cet.ncsa.uiuc.edu/2007/"
  xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <cet:Context rdf:about="tag:cet.ncsa.uiuc.edu,2008:/bean/Context/4429050e-f7b0-
4fa7-b1ee-c007ddee8641">
    <a:hasChildren>
      <rdf:Seq>
        <rdf:_1>
          <cet:Context rdf:about="tag:cet.ncsa.uiuc.edu,2008:/bean/Context/f1c18980-
d469-4d71-9a6e-31a266f876c5">
            <a:hasChildren>
              <rdf:Seq>
                <a:hasProperties>
                  <b:storageTypeMapEntry>
                    <b:storageTypeMapKey>MySQL.user</b:storageTypeMapKey>
                    <b:storageTypeMapValue>[put username]</b:storageTypeMapValue>
                  </b:storageTypeMapEntry>
                </a:hasProperties>
                <a:hasProperties>
                  <b:storageTypeMapEntry>
                    <b:storageTypeMapKey>MySQL.host</b:storageTypeMapKey>
                    <b:storageTypeMapValue>[put host URL]</b:storageTypeMapValue>
                  </b:storageTypeMapEntry>
                </a:hasProperties>
                <a:hasProperties>
                  <b:storageTypeMapEntry>
                    <b:storageTypeMapKey>MySQL.password</b:storageTypeMapKey>
                    <b:storageTypeMapValue>[put password]</b:storageTypeMapValue>
                  </b:storageTypeMapEntry>
                </a:hasProperties>
                <a:hasProperties>
                  <b:storageTypeMapEntry>
                    <b:storageTypeMapKey>MySQL.database</b:storageTypeMapKey>
                    <b:storageTypeMapValue>[put database name]</b:storageTypeMapValue>
                  </b:storageTypeMapEntry>
                </a:hasProperties>
                <a:isType>MySQL</a:isType>
                <dc:creator>
```

```

        <foaf:Person
rdf:about="http://cet.ncsa.uiuc.edu/2007/person/anonymous">

<dc:identifier>http://cet.ncsa.uiuc.edu/2007/person/anonymous</dc:identifier>
        <rdf:type
rdf:resource="tag:tupeloproject.org,2006:/2.0/beans/2.0/storageTypeBeanEntry"/>
        <rdfs:label>Anonymous</rdfs:label>
        <foaf:mbox/>
        <foaf:name>Anonymous</foaf:name>
        <b:propertyImplementationMappingSubject
rdf:resource="tag:cet.ncsa.uiuc.edu,2009:/mapping/http://xmlns.com/foaf/0.1/Person"/>

<b:PropertyValueImplementationClassName>edu.uiuc.ncsa.cet.bean.PersonBean</b:PropertyValueImplementationClassName>
        </foaf:Person>
        </dc:creator>
        <dc:date rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2010-02-05T03:35:15.968Z</dc:date>
        <dc:description/>
        <dc:identifier>tag:cet.ncsa.uiuc.edu,2008:/bean/Context/f1c18980-d469-4d71-9a6e-31a266f876c5</dc:identifier>
        <dc:title>MySQL</dc:title>
        <rdf:type
rdf:resource="tag:tupeloproject.org,2006:/2.0/beans/2.0/storageTypeBeanEntry"/>
        <rdfs:label>MySQL</rdfs:label>
        <b:propertyImplementationMappingSubject
rdf:resource="tag:cet.ncsa.uiuc.edu,2009:/mapping/http://cet.ncsa.uiuc.edu/2007/Context"/>

<b:PropertyValueImplementationClassName>edu.uiuc.ncsa.cet.bean.context.ContextBean</b:PropertyValueImplementationClassName>
        </cet:Context>
        </rdf:_1>
        <rdf:_2>
        <cet:Context rdf:about="tag:cet.ncsa.uiuc.edu,2008:/bean/Context/c67a3cdb-c08c-4289-aef2-6827d0d78c90">
            <a:hasChildren>
                <rdf:Seq
rdf:about="tag:tupeloproject.org,2006:c8991733d11ad088c721e633f774e38edec0363e"/>
                </a:hasChildren>
                <a:hasProperties>
                    <b:storageTypeMapEntry
rdf:about="tag:tupeloproject.org,2006:1f54b2550c47fe349b23567961f120c650556174">
                    <b:storageTypeMapKey>folder</b:storageTypeMapKey>
                    <b:storageTypeMapValue>[data folder location]</b:storageTypeMapValue>
                    </b:storageTypeMapEntry>
                </a:hasProperties>
                <a:isType>HashFile</a:isType>
                <dc:creator
rdf:resource="http://cet.ncsa.uiuc.edu/2007/person/anonymous"/>
                    <dc:date rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2010-02-05T03:35:15.968Z</dc:date>
                    <dc:description/>
                    <dc:identifier>tag:cet.ncsa.uiuc.edu,2008:/bean/Context/c67a3cdb-c08c-
```

```

4289-aef2-6827d0d78c90</dc:identifier>
    <dc:title>HashFile</dc:title>
    <rdf:type
rdf:resource="tag:tupeloproject.org,2006:/2.0/beans/2.0/storageTypeBeanEntry"/>
    <rdfs:label>HashFile</rdfs:label>
    <b:propertyImplementationMappingSubject
rdf:resource="tag:cet.ncsa.uiuc.edu,2009:/mapping/http://cet.ncsa.uiuc.edu/2007/Conte
xt"/>

<b:PropertyValueImplementationClassName>edu.uiuc.ncsa.cet.bean.context.ContextBean</b
:PropertyValueImplementationClassName>
    </cet:Context>
    </rdf:_2>
    </rdf:Seq>
    </a:hasChildren>
    <a:isType>ContentStore</a:isType>
    <dc:creator rdf:resource="http://cet.ncsa.uiuc.edu/2007/person/anonymous"/>
    <dc:date rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2010-02-
05T03:35:15.968Z</dc:date>
    <dc:description/>
    <dc:identifier>tag:cet.ncsa.uiuc.edu,2008:/bean/Context/4429050e-f7b0-4fa7-b1ee-
c007ddee8641</dc:identifier>
    <dc:title>MySQL and Hashfile</dc:title>
    <rdf:type rdf:resource="tag:tupeloproject.org,2006:/2.0/DefaultContext"/>
    <rdf:type
rdf:resource="tag:tupeloproject.org,2006:/2.0/beans/2.0/storageTypeBeanEntry"/>
    <rdfs:label>MySQL and Hashfile</rdfs:label>
    <b:propertyImplementationMappingSubject
rdf:resource="tag:cet.ncsa.uiuc.edu,2009:/mapping/http://cet.ncsa.uiuc.edu/2007/Conte
xt"/>

<b:PropertyValueImplementationClassName>edu.uiuc.ncsa.cet.bean.context.ContextBean</b
:PropertyValueImplementationClassName>
    </cet:Context>
</rdf:RDF>

```